**Denoising Autoencoder**

Zheng Lu (ECE692)

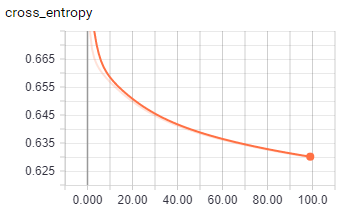
## Task 1:

1. Common settings for experiments of task 1:

* hidden layer size: 1024
* batch size 64
* epochs 100
* learning rate 0.05

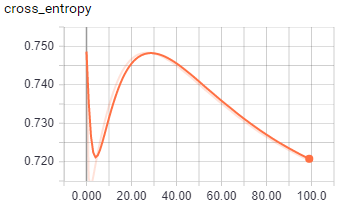
1. Out-of-box (yaldt) experiment running on cifar10:

* corruption module none
* activation function sigmoid
* loss function cross entropy
* stochastic gradient descent without momentum



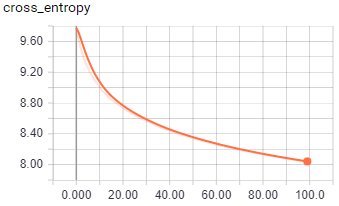
1. Add Gaussian noise as corruption module:

* Same as 1 except:
* Corruption module: Gaussian



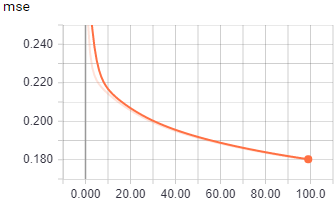
1. Replace sigmoid with ReLu:

* Same as 1 except:
* activation function ReLu



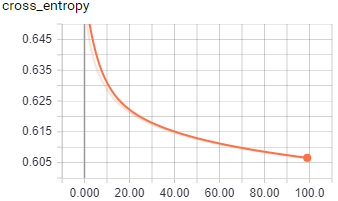
1. Replace cross entropy with squared loss:

* Same as 1 except:
* loss function squared loss



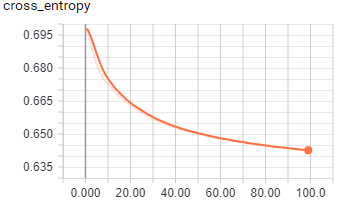
1. Add momentum:

* Same as 1 except:
* stochastic gradient descent with momentum (0.9)



1. Recommended settings:

* Same as 1 except:
* Uniform corruption module, sigmoid, cross entropy and momentum (0.9)



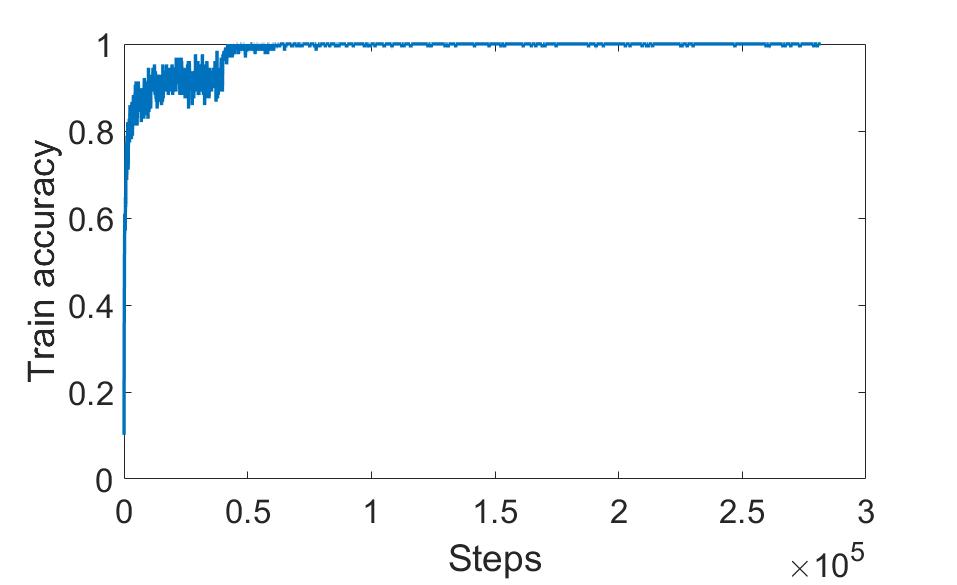
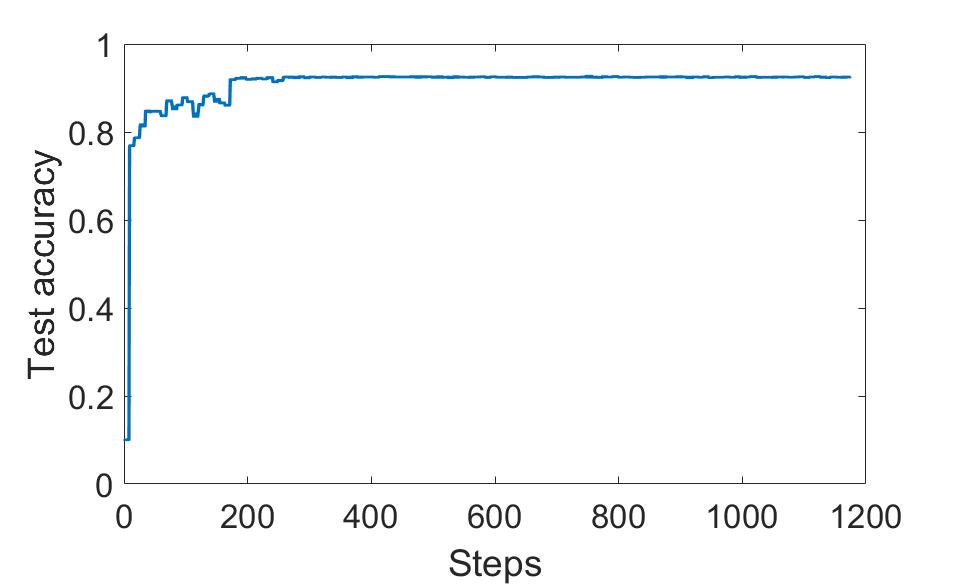
## Task 2:

1. Settings for autoencoder:

* Same hidden layer size, batch size, epochs and learning rate as case 1 in task 1
* Gaussian corruption module
* ReLu activation function
* Cross entropy as loss function
* SGD with momentum

1. Setting for lenet

* Same hidden layer size, batch size, epochs and learning rate as case 1 in task 1
* Gaussian corruption module
* ReLu activation function
* Cross entropy as loss function
* SGD with momentum

1. Training accuracy
2. Testing accuracy (Final testing accuracy 0.929)  
   

## Cost functions and activation functions

1. Quadratic cost function vs. Cross entropy

The quadratic cost function has the problem that when the prediction result is badly wrong, the learning speed is slow as the partial derivatives which are used for learning in back propagation are small.

The cross entropy solves the problem of slow learning when the prediction results is badly wrong. Also, it is non-negative and is close to zero as outputs of the network are close to the correct output. These two factors along with the ability to avoid learning slow down make it very suitable to be a cost function than quadratic cost function.

1. Sigmoid, Softmax, ReLU  
   Sigmoid is used in perceptron to replace step function so that a small change in the network will not cause a dramatic change in the output. This is the basic property that makes learning possible.

Softmax is usually used in the output layer as it has the property that it will generate a set of positive numbers that sum up to 1. In other words, the softmax function can output a probability distribution which is very useful in many cases.  
Sigmoid functions have the saturating problem, that is, the training speed with gradient descent will slow down as the function saturates. ReLU is a non-saturating function which has same good properties as sigmoid function but it won’t slow down the training speed with gradient descent. This property makes it very suitable for deep convolutional neural networks that requires fast training.

## Paper review (ECE692 only)

1. LeCun 1998  
   The paper reviews various methods applied to handwritten digit recognition task. It shows that automated feature generation can outperform features extracted by domain experts. It reviews recent popularized learning from data approach which is an optimization problem based on gradient descent algorithm. Besides that, it also reviews LeNet, a simple example of CNN. There are 4 important ideas in CNN, namely reception field, pooling, shared weights and multiple layers. In this type of neural network, the convolutional layer is to detect features while the pooling layer is to merge similar features into one. With them combined to form one layer of CNN, each of such layer is to reliably detect local conjunction of features from outputs of the previous layer. With these technologies, CNNs have much fewer parameters to train and can potentially achieve similar performance with fully connected multilayer neural networks.
2. AlexNet  
   The available of large scale datasets such as ImageNet, the relatively faster training procedure of CNNs and the computation power increases in GPU make it possible for the existence of a deeper neural network than ever before. AlexNet is the first work that shows deep learning classifiers can significantly outperform shallow classifiers as long as it is properly trained. To facilitate the training procedure, they use non-saturating activation functions to avoid the learning slow down problem and implement the model on multiple GPUs. Besides that, local response normalization and overlapping pooling are used to further improve the performance of the network. To reduce the overfitting problem, the authors create more data for training by use data augmentation and use dropout to prevent co-adaptations of neurons.
3. GoogLeNet

This paper studies another perspective of neural networks, their efficiency. They shows how to increase the depth and width of the network while still using same level of computational budget. The main concern of the paper is how to approximate the optimal local sparse structure of the CNN with the already available dense components. Based on this motivation, they propose the inception architecture which has the “network in network” structure that can effectively increase both the depth and the width of the network.

1. VGGNet  
   This paper propose a CNN architecture that only use 3x3 convolution filters. The paper wants to push the depth of the CNN to its limits by adding more convolutional layers. To make it feasible, they only use the 3x3 kernals, which is the smallest size to capture various directional notions. The reason for using 3x3 kernal is that the reception field of the larger size kernals can be effectively represented as stackings of 3x3 convolutional layers.
2. ResNet

This paper mainly solves one problem: Is learning better networks as easy as stacking more layers. However, to answer such a question, one must conquer the vanishing gradient problem. Without solving this problem, adding more layers may get worse performance. The authors propose a new network framework called residual learning framework that instead of learning the desired mapping, they let the network learning the residual mapping, that is the desired mapping minus input. Such an approach effectively solves the gradient vanishing problem and achieve network depth that has never existed before, i.e., over 1000 layers. Not surprisingly, such deep networks can achieve much better performance than previous networks.